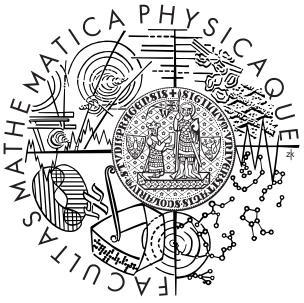
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BACHELOR THESIS



Michal Demin

Robot localization using MEMS sensors

Katedra softwarového inženýrství

Vedoucí bakalářské práce: RNDr.David Obdržálek
Studijní program: programování

2010

Ďakujem svojmu vedúcemu a všetkým, ktorí ma podporovali pri pisani tejto práce.

Prohlašuji, že jsem svou diplomovou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

V Praze dne Michal Demín

Contents

Název práce: Robot localization using MEMS sensors
Autor: Michal Demin
Katedra (ústav): Katedra softwarového inženýrství
Vedoucí bakalářské práce: RNDr. David Obdržálek
e-mail vedoucího: david.obdrzalek@mff.cuni.cz

Abstrakt: Výskyt autonómnych robotov je v súčasnej dobe vo svete čoraz bežnejší. Aby vedeli prežiť, potrebujú presné informácie o svojej polohe. Cieľom tejto práce je navrhnúť a vytvoriť robustnú lokalizačnú knižnicu, ktorá by poskytovala tieto informácie robotom. MEMS senzory sa používajú ako inerciálne snímače, ktoré poskytujú knižnici akceleračné a rotačné dáta. Diskutujeme o rôznych MEMS senzoroch a ich použiteľnosti pre tieto účely. Popisujeme rozličné metódy použité na spracovanie senzorových dát, ako sú: Kalmanov filter, použitý k odstráneniu šumu z meraných hodnôt, zarovnanie senzorov pomocou quaternionov v priestorovej rotácii a z dôvodu kompatibility so vstavanými systémami aj aritmetiku pevnej desatinnej bodky. Zistili sme, že kvôli posunu, ktorý sa objaví po určitom čase, nám môžu gyroskopy podať presné informácie iba na krátku dobu pozorovania. Kvôli nepresnosti snímania zrýchlenia a značnému posunu, sa akceleračné senzory ukázali nepoužiteľné na lokalizačné potreby.

Klíčová slova: MEMS senzory, filtrovanie dát, Kalmanov filter, quaterniony

Title: Robot Localization Using MEMS Sensors
Author: Michal Demín
Department: Department of Software Engineering
Supervisor: RNDr. David Obdržálek
Supervisor's e-mail address: david.obdrzalek@mff.cuni.cz

Abstract: Autonomous robots are becoming more common in this world. To be able to survive, they need precise information about their position. Purpose of this thesis is to design and create robust localization library, that provides such information to the robots. MEMS sensors are used as inertial measurement unit, which provide acceleration and rotation data to the library. We are discussing various available MEMS sensors, and their usability for such task. Described are various methods used to process sensor data such as the Kalman filter used to filter noise from measurements, sensor alignment using quaternions as spacial rotation, and fixed-point arithmetics for compatibility with embedded systems. We found out that gyroscopes can only be used for short term observations, because of their drift. Acceleration sensors proved themselves unusable for localization purposes, because of imprecise acceleration sensing and major drift over time.

Keywords: MEMS sensors, data filtering, Kalman filter, quaternions

Chapter 1

Introduction

Today robots can be found almost everywhere in our lives. Autonomous robots are serving to explore new planets, explore confined spaces, or clean the house. These robot has to be able to do correct decisions on which way to go, these decisions need correct input of the location where the robot is. For large scale global navigation systems have been developed. These systems work great with sky visible, but useless indoors.

The focus of this thesis is to implement robust localization on small scale playfield. The position is reported relatively to the start position. Inertial measurement unit (IMU) is used to provide data to the library. IMU is made of several MEMS sensors, such as accelerometers and gyroscopes. These MEMS sensors can sense slight changes in movement with regard to Earth's position and convert these changes into electrical information. In this thesis we are discussing various obstacles that need to be overcome to get from raw data, provided by the IMU, to the reliable position information, that can be used in higher level software.

MEMS devices have been available for a while on market. But only recent development in technology and price drops have made these available to the wide public. Such sensors are already part of many cell-phones, PDAs and other consumer electronics. They are used to measure the tilt of screen, measure temperature, etc.

This thesis has been divided into 6 different chapters, each is describing different part of development process. In chapter 2 we are introducing MEMS sensors, the technology behind, different types of devices. Most useful sensors for navigation are used in this chaper. Brief concept of the library with selected methods is described in the chapter 3. The functionality and analysis of the selected methods is described in chapter (4). Chapter 5 shows use of the implemented library on working autonomous robot, various moves are evaluated for better understanding.

Chapter 2

MEMS sensors

2.1 Technology

Micro-electro-mechanical system (MEMS) is technology, where mechanical elements, such as sensors, actuators are embedded with common electronic integrated circuit into one package. The electronic part of the device is manufactured using standard process sequences, while the mechanical part of the device is "etched" using micro-machine processes. Size of mechanical components is in range from $100\mu m$ to $0.1mm$ while the sizes of whole devices are usually somewhere between $20\mu m$ to $1mm$.

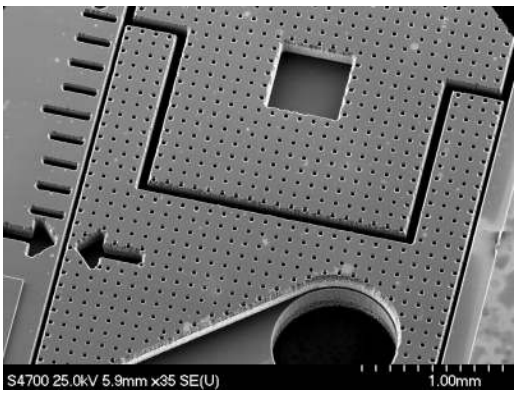


Figure 2.1: MEMS sensor under electron microscope

Embedding the sensors with the electronic chip enables to build system-on-chip devices. The mechanical sensors are used to sense the environmental changes, chemical, thermal, optical, and the electronic part uses the information for fast decision making.

Only small changes need to be done to the actual manufacturing process so relatively low price of the devices is maintained. This allows to use these devices in various consumer electronics like cell phones to sense the tilt of the screen, laptops to protect fragile hard-drives from shocks, hobby aircrafts, even medical micro-robots.

More information on these sensors can be found [1].

2.2 Available devices

There are various MEMS devices available for example: inertial sensors, accelerometers and gyroscopes, pressure sensors, displays, or even ink-jet printers. These devices found ways into various fields like chemistry, biology, medicine, communication, environment sensing, etc. Their potential has not been fully explored so far, and their use is practically unlimited. Many project exist now days to create miniature robot to save explore human bodies from inside, create devices for DNA sequencing, that can be done more efficiently or biochips that will detect hazardous chemicals in air or in blood.

Despite the large selection of available MEMS sensors, the most interesting for our purpose are Accelerometers, and Gyroscopes. These sensors provide us with relative information about the "movement" of the device, so called Inertial Measurement Units or IMU. Accelerometers are measuring acceleration with relation to the Earth's gravity. They are implemented as small "springs" that stretch due to the effect of the gravitation force. The amplitude is sensed by included electronic device and provided as output for further processing by application.

Gyroscopes are more complicated devices. Gyroscopes contain vibrating structure, that tends to vibrate in the same plane as it's support plane is rotated. Upon rotation the Coriolis force can be measured by support electronic. This force is proportional to the amount angle speed applied to the sensor.

As these devices are mechanical care has to be taken when handling them. Stressing the mechanical structure by sudden temperature changes, or excessive vibration, mechanical shock can cause the MEMS structure to become unstable or damaged. This can lead to major change in mechanical properties, making the sensor less immune to noise, and issuing extra offset to the output data.

For our thesis we have chosen sensors that are available in the local store. For gyroscope ADIS16250 manufactured by Analog devices and accelerometer LIS3LV02. All the properties of these chips can be obtained from datasheets: ADIS16250 [3] and LIS3LV02 [2].

Chapter 3

The Concept of Library

In this chapter we are describing concept of the library that processes the MEMS data in order to provide reliable position. The position as we refer to is state of watched object in certain time. This state expresses following parameters: position, speed, acceleration and velocity in each axis (x,y), angle and angular velocity. These parameters are used to estimate next position of the observed object. Calculations are done as soon as new data from the sensors is available, and the time difference between each measurement should be constant. This time should be known before initialization of the library.

We assume that the measurements are available to the library, and various problems with hardware dependent setup are not aim of this thesis. The MEMS sensors (see 2) provide analog data, that needs to be processed in AD converter. Analog signal is very sensitive to external noise, such as EMI - Electro-magnetical interference. These interferences are often higher frequency than of our measured data or have some predictable pattern, so we can easily filter them out. Used filtering method is discussed in in section 3.2.

Once the filtered data is available some normalization and calibration data has to be applied (Section 3.3). The reason for toing so is that each sensor possess different electro-mechanical properties. These properties are defined by manufacturing process errors and cannot be predicted. We need to be sure that the scale of the data is defined and known by the upper level of this library. Each sensor needs to be separately calibrated. This calibration does not change by time, but extreme temperature changes can cause some slight offsets in measurement.

Further calibration includes alignment of the accelerometer axes with the inertial system axes. This process is described in the section (3.4).

Further data processing includes integration of various sensors measurement data (angular speed, and acceleration) to obtain other position parameters like: speed, position, and angle. (Section 3.5)

3.1 Interfaces and sensor definition

Each sensor is defined by series of parameters, these parameters need to be determined before we can use the library reliably. As each sensor has different properties, these parameters have to be determined for each sensor independently.

- Normalization and calibration data
- Statistical model of the output data, used for filtering
- Alignment data
- Time difference between measurements

As MEMS sensors properties change very little over temperature range, and over time these parameters can be considered static. Note should be taken, that sensors are sensitive to shocks (either temperature or mechanical). For optimal performance calibration process should be repeated before the library usage.

3.2 Filter

We are using Kalman filter to remove noise from the measured data. The Kalman filter is a mathematical method named after Rudolf E. Kalman. It estimates real measurement value by predicting and estimating the value of noise and producing weighted average of the predicted and measured value. The "basic" model of Kalman filter estimates Gaussian white noise in the process and in the measurements. These noises are assumed to be independent, with normal distribution - white noise. The main parameters of the Kalman filter are the process noise covariance, and measurement noise covariance. Both of these parameters are stored in the sensor definition structure (3.1).

In our model we are assuming that each signal being filtered is independent, and that we are using the Kalman filter to remove measurement noise introduced before data by data capture.

The measurement covariance can be computed by computing variance while the process is in calm state and calculating the variance of the input signal using statistical calculations. The process covariance is harder to determine, and reasonably good results can by obtained by empirically determining the value.

3.3 Calibration

The filtered measurement data coming from the sensors is raw, in no particular unit. It needs to be scaled to express some units that can be used in further processing. We are scaling the data to represent units in SI (m/s^2 in case of accelerometer, and deg/s in case of gyroscope). The data we are capturing is processed through a linear equation where the raw data gets scaled and zeroed to represent value in defined units.

The calibration of accelerometer is using Earth's gravity as reference. By finding the extremes (maximum and minimum) for each axis we are able to compute both offset and scale.

For gyroscope we need the zero rotation as reference, which tells us the offset. Next we need to rotate the sensor by known angle. Which is than fed back to the calibration process to determine the correct scale.

3.4 Alignment

In real environment we were unable mount the accelerometer aligned exactly with with Earth gravitation. There is almost always slight rotation of our system. The solution is to rotate our system in software after we have normalized data available. Many methods exist to apply spacial rotation like Euler angles, or quaternions.

We have selected quaternions for this job. Quaternions have found their way to many fields like computer graphics. They are better in some ways than Euler angles. One of these is that they do not suffer from "Gimbal Lock". Rotating the system with Euler angles means: If we have two systems xyz and XYZ, the XYZ as origin, we are able to reach any target frame by composing three rotations. We start with xyz and XYZ coinciding. We rotate the xyz-system about z-axis by angle α. Then rotate xyz-system about x-axis by angle β. And finally rotate the xyz-system about the new z-axis by angle γ.

Rotation using quaternion can be viewed as rotation of plane around certain vector by angle. This method is slightly less intuitive, but in the end uses less operations than calculating using Euler angles. pose.

3.5 Further processing

Once our data has been calibrated, normalized, and aligned they can be further processed. Simple algorithms are used to calculate the missing position parameters - velocity, position and angle.

Gyroscope data is used to directly calculate the angle by integrating the measured data. As we are doing calculations in discrete time, integration means summing small changes of angle differences. $\sum \omega dt$, where ω is the angular speed reported by gyro sensor and dt is the time difference between consequent measurements.

The computed angle is saved in the position structure and further used to calculate the effect of rotation on the acceleration and velocity. Accelerometer data is rotated using simple transformation matrix to rotate the acceleration vector to express the real acceleration of the object. The resulting data is integrated, in similar fashion as the gyro data, to get the velocity vector. The velocity vector is then integrated to get delta position coordinates.

In case some measurement information is missing or the user possesses higher accuracy data, we have provided interface that will allow to override the measurements. This would help to reset the drift that MEMS sensors have.

3.6 Setup for our robot - example

The primary use of this library is on autonomous robot. We have equipped our robot with inertial measurement unit - one three accelerometer with $\pm 2g$ sensitivity [2], and one yaw gyroscope with $\pm 300°/sec$ sensitivity [3]. Data from both sensors are provided at fixed rate every $27ms$ or $37Hz$. This data is accessed by simple reading of file from the programming language point of view and all the measurements are

provided in raw format. Output is proportional to the applied force/angular speed, and with no known scale.

This sensor is used to test each feature programmed in this library, such as statistical noise error, developed calibration methods. This sensor is also used to provide measurements of various robot movements on the play-field. The result and discussion related to this testing is in chapter (5).

Chapter 4

Functionality analysis

4.1 Data filtering

To filter out any measurement and environmental noises, we have implemented the Kalman filter. The Kalman filter estimates state $x \in R^n$ of a discrete-time controlled process given by a linear stochastic difference equation

$$x_k = Ax_{k-1} + Bu_{k-1} + w_{k-1} \tag{4.1}$$

and a measurement $z \in R^n$

$$z_k = Hx_k + v_k \tag{4.2}$$

Where variables w_k and v_k represent process and measurement noise. We assume that these variables are independent, with normal probability distributions

$$p(w) \sim N(0, Q) \tag{4.3}$$

$$p(v) \sim N(0, R) \tag{4.4}$$

We assume that the Q, process noise covariance, and R, measurement noise covariance, are constant. Matrix A (n x n) gives the relation between the state in time $k-1$ and k. Matrix B (n x l) relates the control input vector $u \in R^l$ to state x. H is the observation model mapping the true model into observation model.

In our system only observations are made, so we can omit the control input. Giving us simpler equation

$$x_k = Ax_{k-1} + w_{k-1} \tag{4.5}$$

Kalman filter [4] estimates the process at some time and then obtains feedback in form of measurement. This leads to two phases that Kalman filter uses: Predict and Update. In Predict phase filter predicts the next state towards current. In this phase it also predicts the error covariance for next step. The update phase is responsible for the feedback, the measurement information at current time-step is used to improve the prediction in next step. These two phases can be described by the following equations:

For Predict phase:

$$\hat{x_k} = A\hat{x_{k-1}} \tag{4.6}$$

$$P_k = AP_{k-1}A^T + Q \tag{4.7}$$

For Update phase:

$$K_k = P_k H^T (HP_k H^T + R)^{-1} \tag{4.8}$$

$$\hat{x_k} = \hat{x_k} + K_k(z_k - H\hat{x_k}) \tag{4.9}$$

$$P_k = (I - K_k H)P_k \tag{4.10}$$

where P_k is the error covariance in time k, $\hat{x_k}$ is the predicted state in time k, K_k is called Kalman gain.

The Predict phase predicts the current state using the previous state and the model described by matrix A (4.6), and calculates the error covariance for the current time-step (4.7). Initial error covariance, for time $k = 0$, should be set to express the initial state of the model.

The Update phase computes the Kalman gain K_k (4.8). Kalman gain purpose is to minimize error covariance P_k. The next step is to incorporate the actual process measurement into the predicted state(4.9). And finally update the estimate error covariance for this time-step (4.10).

As the variables Q and R are constants the Kalman gain and estimate error covariance will settle on constant value after some time.

The filter is implemented by directly rewriting the equations mentioned into programing language. We assume that the process is observed at some constant rate that is known prior to running the filter, and measurements are passed to this function as parameters. This function predicts the next state from the input parameter and previously saved state variables and updates variables with new state accordingly. All the state variables are kept inside an compact structure for manipulation with the data. These parameters are $\hat{x}$, P, Q, R.

We are using slightly simplified version of the described equations. All the vectors have 1 element, and all matrices are $1x1$ - scalars. The relational matrix, A, is equal to $A = 1$, and observation model mapping is $H = 1$. The resulting equations are:

For Predict phase:

$$\hat{x_k} = \hat{x_{k-1}} \tag{4.11}$$

$$P_k = P_{k-1} + Q \tag{4.12}$$

For Update phase:

$$K_k = P_k(P_k + R)^{-1} \tag{4.13}$$

$$\hat{x_k} = \hat{x_k} + K_k(z_k - \hat{x_k}) \tag{4.14}$$

$$P_k = (I - K_k)P_k \tag{4.15}$$

This simplified model is enough to estimate the measurement error and to filter out any noise that is caused by external interference such as EMI.

Tunning of the two error covariance variables is important to obtain optimal result. First the measurement error covariance can be determined by observing the process in calm state. This way the process noise and variation should be eliminated, leaving only the measurement error. Then we need to determine the variance of the error noise. This can easily be done on larger set of captured data, by calculating the mean value and applying the statistical formula for variance (4.16).

$$Var(x) = \sum p_i(x_i - \mu)^2 \tag{4.16}$$

where p_i is the probability mass of x_i, μ is the mean value. The resulting variance is fed into the filter in the initialization stage and is assumed constant throughout whole object observation.

The determining the process error covariance is more difficult as we do not have ability to directly observe the estimated process. This value can be determined in trial-error fashion. Even higher than optimal values (more uncertain) of Q can produce acceptable results. If the value of Q is too high from optimal value, the filter might become less responsive to the values measured.

We have ran some tests on out sensors. Measurement error and process error have been estimated to show the capabilities of this filter. The process noise covariance is $Q = 0.0002$ and measurement error covariance is $R = 0.073$ for out accelerometer. First measurement is used to initialize the error covariance P_0. This helps the filter to reach the correct estimate sooner. The performance of the Kalman filter can be seen in the figure (4.1). The red line represents raw measurements, and green line represents the estimate of the Kalman filter. As we can see most of the noise has been removed and sudden changes need more time to propagate to the Kalman filter's estimate. The error between measured values and estimated values is shown in figure (4.2).

We have also tested the performance of the Kalman filter on out gyro sensor. Different values of measurement error covariance and process error covariance have been estimated $R = 0.25$ and $Q = 0.1$. As we can see most of the high frequency noise has been removed from the signal. The designed Kalman filter works as expected once it is provided with correct statistical information about the noise present. Further evaluation of this filter is in the chapter (5), with proper calculation of the measurement errors of the sensors.

4.2 Sensor calibration routines

We are trying to scale the data that comes out of the sensors to be expressed in some known units, that can be further processed - m/s^2 for accelerometer and $^\circ/sec$ for gyroscope. We assume that the sensor output is linear. According to the datasheets reviewed there is typically $\sim 1\%$ of nonlinearity on full scale for each sensor. This nonlinearity can be neglected. We are fitting the sensor output values into a linear equation - for accelerometer (4.17)

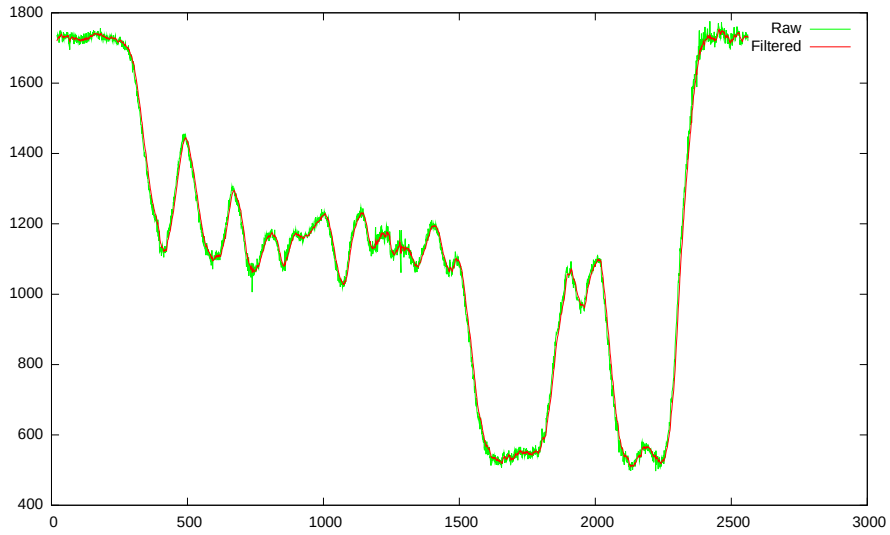


Figure 4.1: Sample Kalman performance on X axis of accelerometer (raw and filtered)

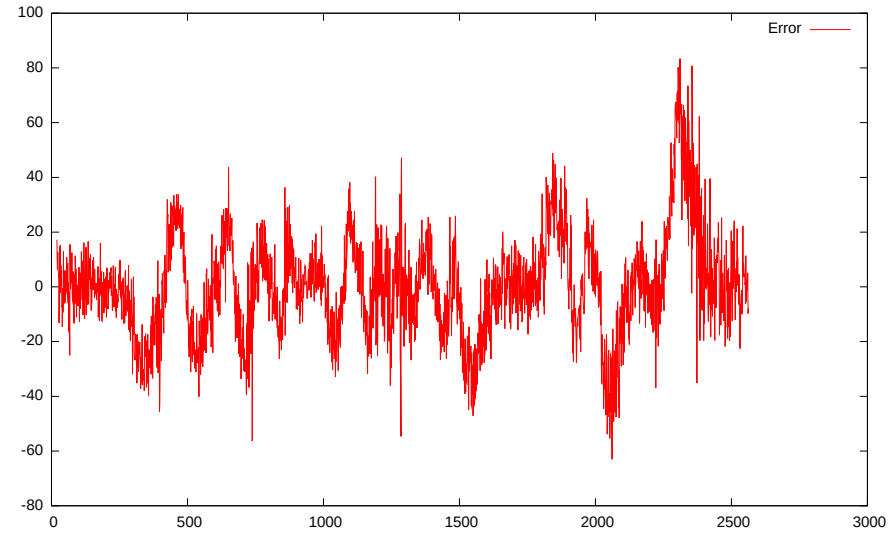


Figure 4.2: Calculated error between raw and filtered value

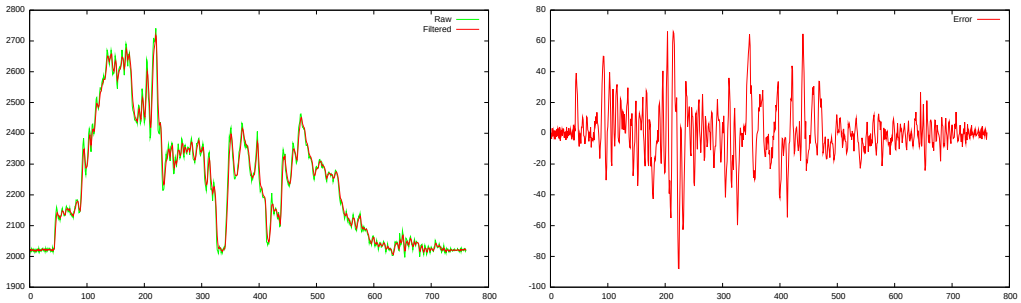


Figure 4.3: Gyroscope measurement and error

$$a_t = \alpha a'_t + \beta \tag{4.17}$$

where the a_t and a'_t are the true acceleration and measured acceleration respectively in time t, α and β are the two coefficients, α being the scaling factor, β being the zeroing offset.

Using Horner scheme we can efficiently evaluate the given polynomial once the calibration coefficients are available. This method can be extended to support any number of coefficients for further possible enhancements.

The accelerometer sensor can be calibrated using the following simple steps. We always need a reference point to be able to determine the coefficients correctly. Our reference force for accelerometer calibration will be the Earth's gravitation force - $1g$. For every linear equation we need at least 2 points to be able to determine the coefficients. This reference force is used to obtain two measurements, one being maximal, and one minimal - corresponding to $+1g$ and $-1g$ respectively. Rotating the accelerometer about each axis x, y, z will lead to 6 extreme values - 3 maximums, and 3 minimums (see fig 4.4).

Next step is to calculate the coefficients of the linear equation that is defined by the two points. This calculation is done for each asix separately.

$$offset = \frac{max + min}{2} \tag{4.18}$$

$$center = max - offset \tag{4.19}$$

$$\alpha = abs(\frac{1}{center}) \tag{4.20}$$

$$\beta = offset * \alpha \tag{4.21}$$

First we compute the offset (4.18), which is the distance between $0g$ and $1g$. Next we find the center value (4.19). This center is used to compute the actuall scale α (4.20) and finally α is used to scale the offset to represent the β (4.21).

These coefficients are stored in the sensor definition, and should change very little over time. Temperature changes can cause the offset to change slightly. Calculated coefficients can be seen in following table

Axis	α	β
x	0.00161	-1.822
y	0.00163	-1.874
z	0.00136	-1.746

These are valid only for out experimental inertial measurement units.

In figure (4.4) we have raw measurements, while rotating the sensor around each axis. All the maximum values can clearly be seen in first thousand samples. Between 1000 and 2000 we have the minimum values for each axis. These values have been already filtered. Using these extreme values we have calculated coefficients for each axis. The same data set has been "calibrated". This way we can check the correctness

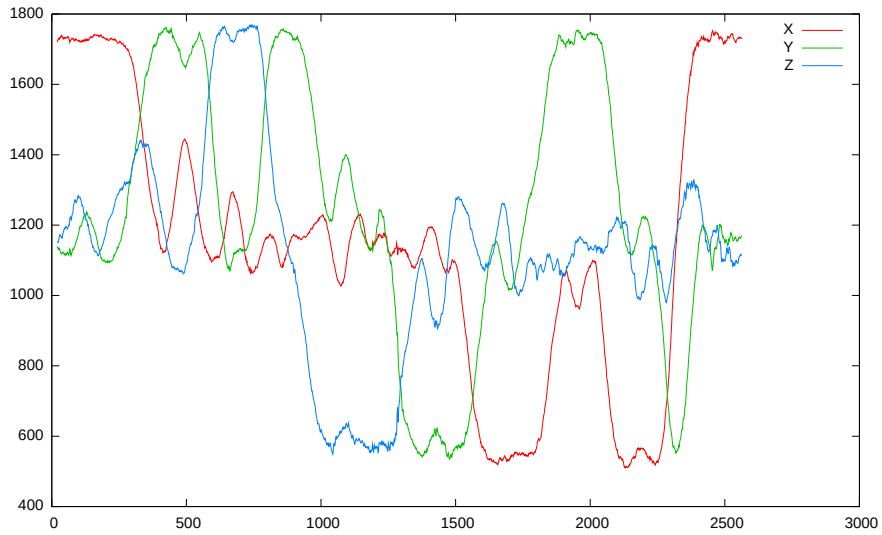


Figure 4.4: Filtered measurements

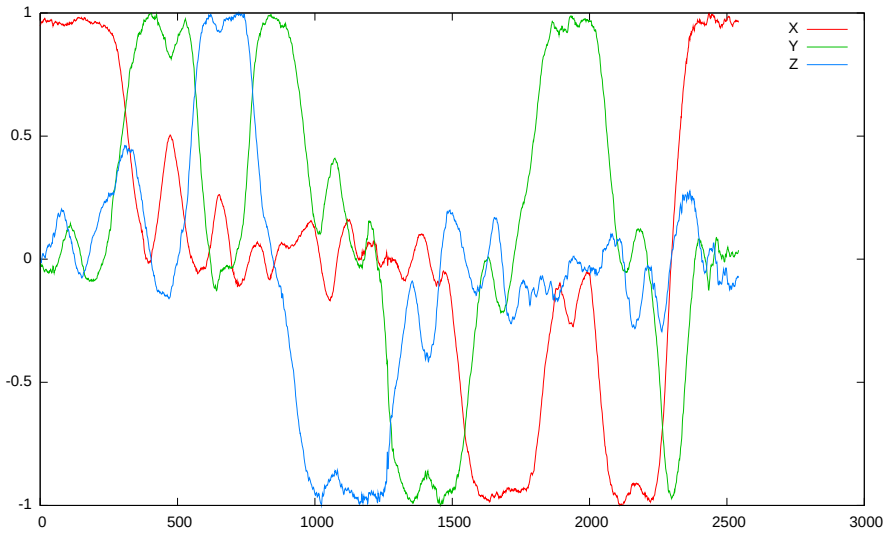


Figure 4.5: Calibrated measurements

of the developed algorithm. Calibrated dataset can be seen in figure (4.5). We can clearly see that the y axis is ranging from -1 to 1 and all the sensors are within the range at all time.

As the gyroscopic sensor is much different from accelerometer different calibration procedure needs to take place. We are using the same scheme as in previous case. We need to find two coefficients (γ and δ) for linear equation (4.22):

$$\omega_t = \gamma\omega'_t + \delta \tag{4.22}$$

where ω_t and ω'_t is true angular velocity and measured angular velocity respectively in time t, γ is the scale, δ is the zero offset.

As the data coming from the sensor contains noise we need to be working with already filtered data. We need to apply the Kalman filter described in section 4.1. After the data is clean we can start calibrating.

First of all we need to determine the zero offset. The zero offset of the gyro sensor can be observed by observing the measured values of stil sensor. This value, δ', is not the final zero offset. The final δ' is the δ' that needs to be scaled by the scaling factor γ - (4.23). The measured values from the gyroscope can be seen in figure (4.6).

$$\delta = \frac{\delta'}{\gamma} \tag{4.23}$$

To determine the correct γ we need some reference angular speed. Straight forward approach can be taken to calculate the scale. We apply the angular speed to the sensor, take measurement, and use equation (4.24) to calculate the scale.

$$\gamma = \frac{\omega}{\omega'} \tag{4.24}$$

As such device is not available in most amateur conditions we have come up with different approach. We turn the sensor by know angle ϕ, record the measurements, and apply discrete integration. The scale can be calculated from given equation by appropriate substitutions. From physics we know that angle can be calculated from known angular speed using equation (4.25).

$$\phi = \omega * t \tag{4.25}$$

where *phy* is angle, ω is angular velocity, and t is time. This formula is, however, valid only for constant angular speed, that we can not maintain by hand. So more complex form (4.26) has to be used.

$$\phi = \int \omega_t dt \tag{4.26}$$

As we are dealing with discrete time system, the (4.26) formula can be rewritten as (4.27).

$$\phi = \sum \omega_t dt \tag{4.27}$$

The actual ω_t is not known, but we can use formula (4.22) with the offset δ from (4.23).

$$\phi = \gamma \sum(\omega'_t - \delta')dt \tag{4.28}$$

$$\gamma = \frac{\phi}{\sum(\omega'_t - \delta')dt} \tag{4.29}$$

The final formula (4.29) is now used to compute the correct scale. Afterwards γ is used to calculate the correct value of offset δ using formula (4.23).

We have captured some data while the sensor has been in stil position, and another data while full rotation (360°) has been applied to the sensor. This data has been filtered and can be seen in figure (4.6). We were unable to turn the sensor with constant angular speed as can be seen from the figure - The red line is not straight. The green line in figure (4.6) represents the zero offset. After taking the zero offset and using the method described we have calculated the coefficients. These coefficients have been determined to be $\gamma = 0.167$ and $\delta = -338.42$. We have applied these coefficients to the previous dataset and the result can be seen in figure (4.7). Red line represents the actual angular velocity, added blue line represents integrated angle. As we can see the results are as expected.

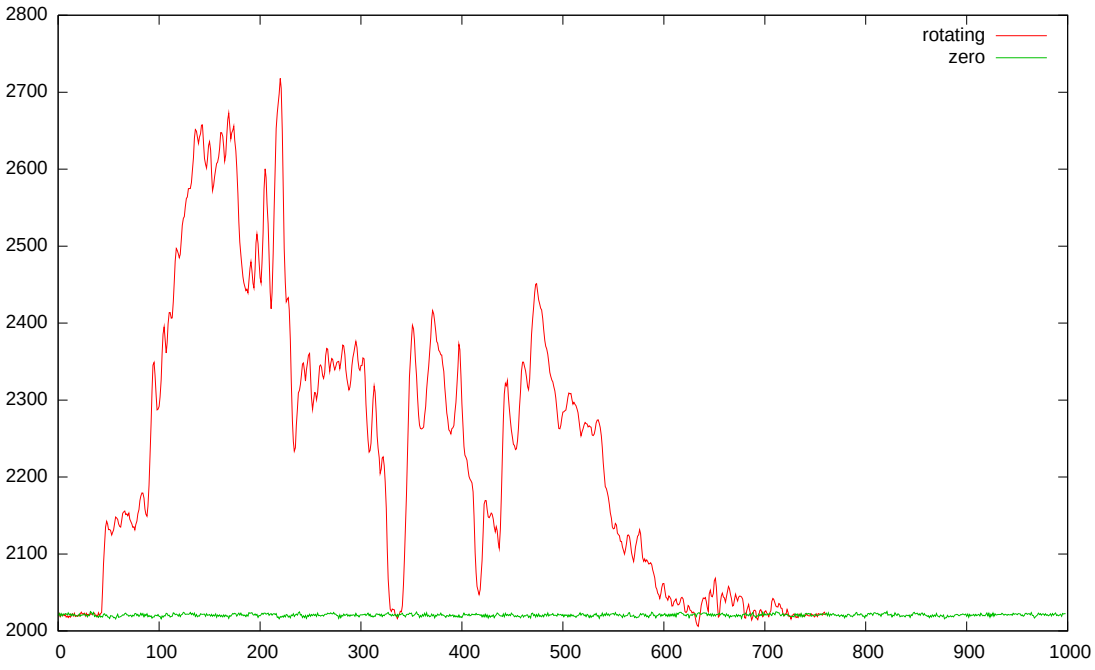


Figure 4.6: Gyroscope during full turn - filtered data

4.3 Alignment routines

As the manufacturing process and the way we attach the accelerometer to the observed device is not perfect, we need a method to align the z-axis with the gravitation

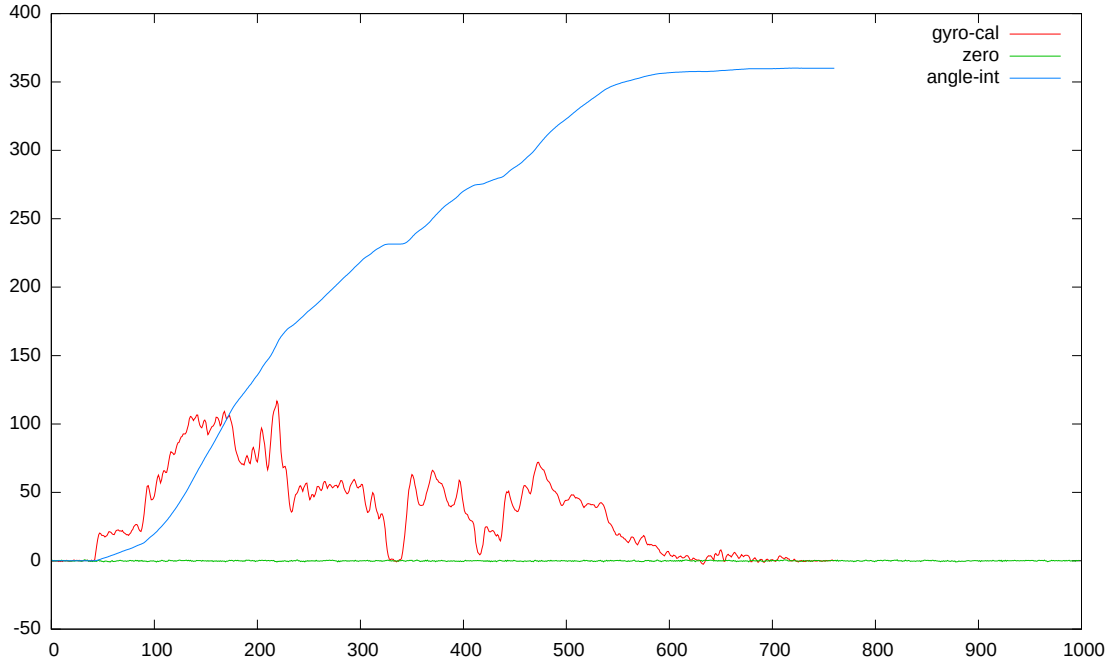


Figure 4.7: Calibrated gyroscope during full turn with integrated values

force, to minimize cross axis measurements. For this purpose we are using quaternions. Quaternions have been widely used for computer graphics, as they do not lead to complications like "Gimbal lock".

Quaternions were conceived by Hamilton in 1800s as extended complex numbers, $w + ix + jy + kz$, with $i^2 = j^2 = k^2 = -1$, $ij = k = -ji$, and real numbers w, x, y, z. We are using property described in Theorem 1 in [5]. If we take quaternion $q = cos\phi + usin\phi$ such that $norm(q) = 1$, the q acts as rotation around axis u by angle 2ϕ

We start by taking two vectors v_1 and v_2, one which correspond to the Earth's gravity force $(0, 0, -1)$ and one of our measurement in stil condition. We are trying to determine the half angle of these two vectors ϕ and vector about which we need to rotate to align these vectors. We start by computing the bisector of the angles between these two vectors. The bisector can by computed just by adding the 2 vectors.

$$b = v_1 + v_2 \tag{4.30}$$

Afterwards we need to normalize the bisector. We obtain dot product of v_1 and bisector b. The normalization ensures that the result is the cosine of the angle between two vectors.

$$v_1 \bullet b = |v_1||b|cos\phi \tag{4.31}$$

The dot product is actually the real number w of the quaternion, that we are trying to determine. The next part is the vector x, y, z. This vector can be obtained by computing cross product of vectors v_1 and bisector b. The cross product is a vector that is perpendicular to the plane defined by the two vectors. The cross product is

computed by formula 4.32.

$$v_1 \times b = |v_1||b|sin\phi \tag{4.32}$$

Both vectors v_1 and b have been normalized, the cross product is the x, y, z vector of the quaternion. This quaternion is normalized and we can use its rotational properties to rotate our coordinate system. To apply the rotation to vector v, we need to compute the product $qvq^{-}1$, which equals to matrix M.

$$M = \begin{vmatrix} 1-2(y^2+z^2) & 2(xy-wz) & 2(xz+wy) \\ 2(xy+wz) & 1-2(x^2+z^2) & 2(yz+wx) \\ 2(xz-wy) & 2(yz+wx) & 1-2(x^2+y^2) \end{vmatrix}$$

This matrix can be prepared during the calibration stage and saved with the calibration data. This leads to more data to be stored, but also saves some operations that would be needed to prepare the matrix. The measured vector is then multiplied by this rotation matrix to get true values. Similar method has been described in [6].

4.4 Fixed point mathematics

This library has been designed to be multi-platform compatible, with embedded computing in mind. Embedded systems do not always posses the capability to do floating point arithmetic calculations. For example high speed digital signal processors are optimized to work with integer instructions, and do not provide floating-point operations at all. Instead floating-point operations are emulated using the integer operations leading to much slower emulation. Fixed-point arithmetics has been designed in favor of slower floating point emulation.

Fixed point arithmetics is done as normal integer arithmetics. For example number 12.34 would be stored as number 1234 with scaling factor of 100. This factor is usually a power of 2 for computer arithmetics for easier implementation. This factor is chosen by designer at compile time, and depends on the specific application and precision expected.

We have implemented fixed point arithmetic functions on 32bit integer variables, with 64bit accumulator. The accumulator is needed with operations like multiply, divide and square-root. Operations like add and subtract are easily implemented using the add and subtract integer operations, as both of the operands are already scaled. As can be seen in (4.33) and (4.34).

$$\frac{op_1}{scale} + \frac{op_2}{scale} = \frac{op_1 + op_2}{scale} \tag{4.33}$$

$$\frac{op_1}{scale} - \frac{op_2}{scale} = \frac{op_1 - op_2}{scale} \tag{4.34}$$

Multiply and divide operations are a bit more complex. The result after multiplying two 32bit numbers will be 64bit in worst case. The whole multiplication takes place in the 64bit accumulator. Result of multiplication is then truncated to fit into the 32bit working space, by taking the middle part, defined by the scaling factor. 4.35

$$\frac{op_1}{scale}\frac{op_2}{scale}scale = \frac{op_1 op_2}{scale} \tag{4.35}$$

Division is implemented by first scaling the first operand into 64bit accumulator. And dividing this accumulator by the second operand using the integer division. Final result is calculating by scaling the divided value. 4.36

$$\frac{(\frac{op_1}{scale}/\frac{op_2}{scale})}{scale} = \frac{op_1/op_2}{scale} \tag{4.36}$$

As we are doing multiplication and division on double precision accumulator, we need to take care of proper rounding before we truncate to single precision variable. As the scale factor is power of two, the actual scaling can be implemented as bit-shift, whose performance is superior to integer division.

Square-root function that is needed in (4.3) has been implemented according to [9]. This is actually square-root function for use with integer values, where we need to apply the square-root function to the scaling factor as well (4.37).

$$sqrt(\frac{op}{scale}) = \frac{sqrt(op)}{sqrt(scale)} \tag{4.37}$$

Further operations include multiplication and division of fixed-point numbers by integer. To implement these operations, classical integer operations can be used, and no further scaling is necessary as can be seen in (4.38) and (4.39).

$$\frac{op_1}{scale}op_2 = \frac{op_1 op_2}{scale} \tag{4.38}$$

$$(\frac{op_1}{scale}/op_2) = \frac{op_1/op_2}{scale} \tag{4.39}$$

To add and subtract integer number from fixed-point numbers, we need to scale the integer variable first, and that would lead to similar situation as in formula (4.33) and (4.34). More information about fixed-point arithmetics can be found in [7], [8].

4.5 Further processing

Further data processing includes integration of acceleration and angular rate data to obtain not know parameters. The remaining parameters are computed using discrete time integration. First we calculate the difference either velocity, position, angular and then we apply this difference to the existing parameters. The integration its self can be described by the formula (4.40) for acceleration. Similar formula is used to calculate actual angle.

$$\sum_{i=0}^{n} a_i dt \tag{4.40}$$

We are assuming that the input data for the integration have been previously filtered and calibrated by methods described earlier. We are trying to integrate acceleration twice to get position. Any amount of noise in the input data will most likely destroy our data after second integration, as we are only able to measure data with amplitude higher than actual noise.

4.6 Library implementation

All of the methods described have been implemented in C language. The implementation was done by coding formulas given into the programming language. The library is divided into different modules, each module contains one functionality (filter, normalization, alignment, mathematics etc).

Each sensor has own structure for calibration coefficients. Accelerometer is using normalization coefficients, alignment, and filter. The gyroscope sensor is using normalization, and filter. There is no alignment done for the gyroscope sensor. Every sensor structure needs to be initialized by user to ensure correct behavior of the library.

Special care was taken to implement the library with the provided fixed-point mathematics. All formulas are written with fixed-point math functions. During compile time, user can choose which mathematics is compiled - either floating-point, or fixed-point. Several macros have been provided to help users define constants in fixed-point notation.

Chapter 5

Test run on Robot

Our motion-sensing hardware is based on 2 MEMS sensors along with micro-controller. These sensors are:

- accelerometer LIS3LV02DL [2] made by Freescale semiconductor
- gyroscope ADIS16250 [3] made by Analog devices

Analog-digital conversion is already integrated in these sensors. Both of these sensors provide 12bit resolution. The accelerometer has range of $\pm 2g$, resolution is about $0.001g$. Gyroscope has range of $\pm 300°/sec$, with resolution of about $0.15°/sec$. Both of these sensors have good noise immunity as the analog part is already integrated.

Our software is able to sample these sensors every $27ms$ or at rate of $37Hz$. This should be sufficient for most of slow and medium speed movements. High-speed movements could introduce more errors to the actual integration process and additional offsets in calculations.

5.1 Accelerometer calibration

The accelerometer calibration process includes the following:

- Find maximum and minimum for each axis
- Calculate correct scale and offset
- Calculate rotational matrix
- Determine the measurement covariance

The maximum and minimum is found by rotating the accelerometer around each axis. This uses the gravitation as reference value. We calculate the correct scale and offset using the formulas described earlier and store them in sensor structure.

Alignment of the accelerometer is done with the accelerometer sensor attached to our robot. The sensor should be placed in its final position.

Measurement covariance is done by observing the sensor in stil position. At first we capture set of data (e.g 500 samples), then the mean and variance is calculated. The variance is used as initialization value to the Kalman filter.

We have created program that does the described process. This process cannot be done automatically and requires user input. We have recorded out calibration process and the values are shown in figure (5.1). Three different datasets are graphed each corresponds to different axis. The samples have been taken at $x = 50$ and $x = 200$ for the X-axis, $x = 350$ and $x = 420$ for Y-axis, and at $x = 550$ and $x = 630$ for Z-axis. Values from $x = 800$ have been taken as input for the alignment and for calculating the measurement error covariance.

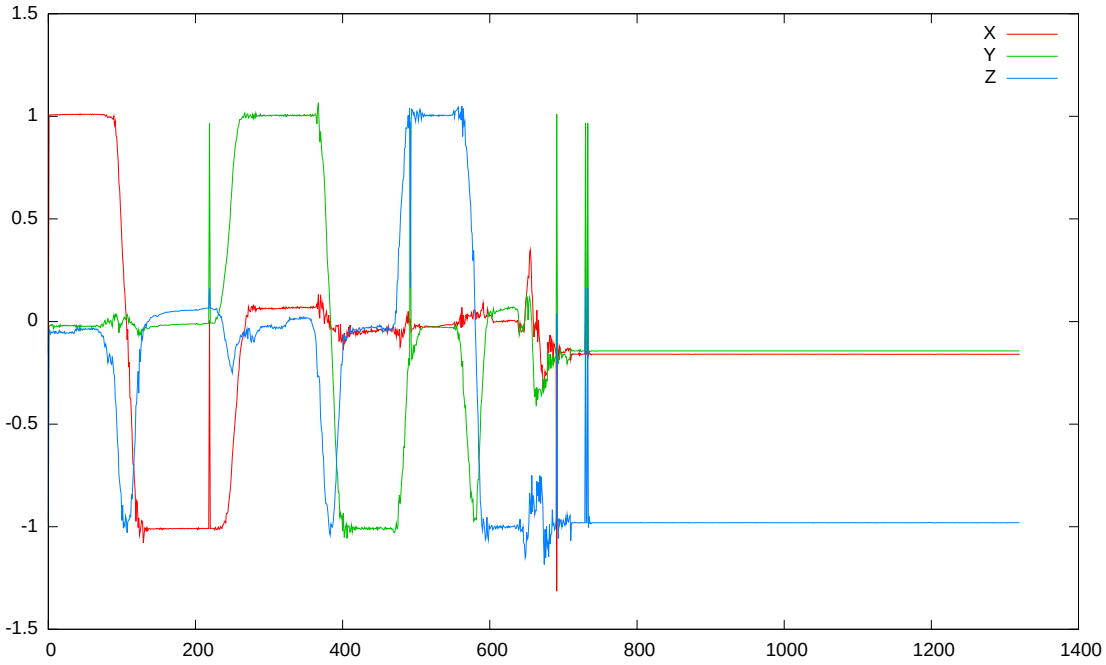


Figure 5.1: Calibration data for accelerometer

The calibration coefficients calculated for out sensor can be seen in following table:

Axis	α	β
x	0.000062	0.004183
y	0.000061	-0.004435
z	0.000061	0.027318

5.2 Gyroscope calibration

For calibrating gyroscope sensor following needs to be done:

- Calculate mean value
- Determine the statistical error

- Calculate scale

Similar as in accelerometer calibration, we capture a set of samples (e.g. 500). These samples are used to determine the mean value and to calculate the variance of the error. This variance is fed to the initialization of Kalman filter. The mean value is also the zero offset of the sensor.

Scale of the gyro sensor is determined by doing full turn (360 degrees) and integrating the values using the formula provided in section (4.2).

In figure (5.2) we have graphed calibration process of the gyroscope, first ~ 500 samples are used to calculate the mean and measurement error variance, after that we have turned the sensor by hand by full turn. The green line represents integrated angle and red line represents angular speed. The sensor was turned by hand so the angular speed could not have been maintained constant. Values in the figure have already been scaled to represent real units.

Coefficients calculated are $\gamma = 0.147039$ and $\delta = -3.943297$.

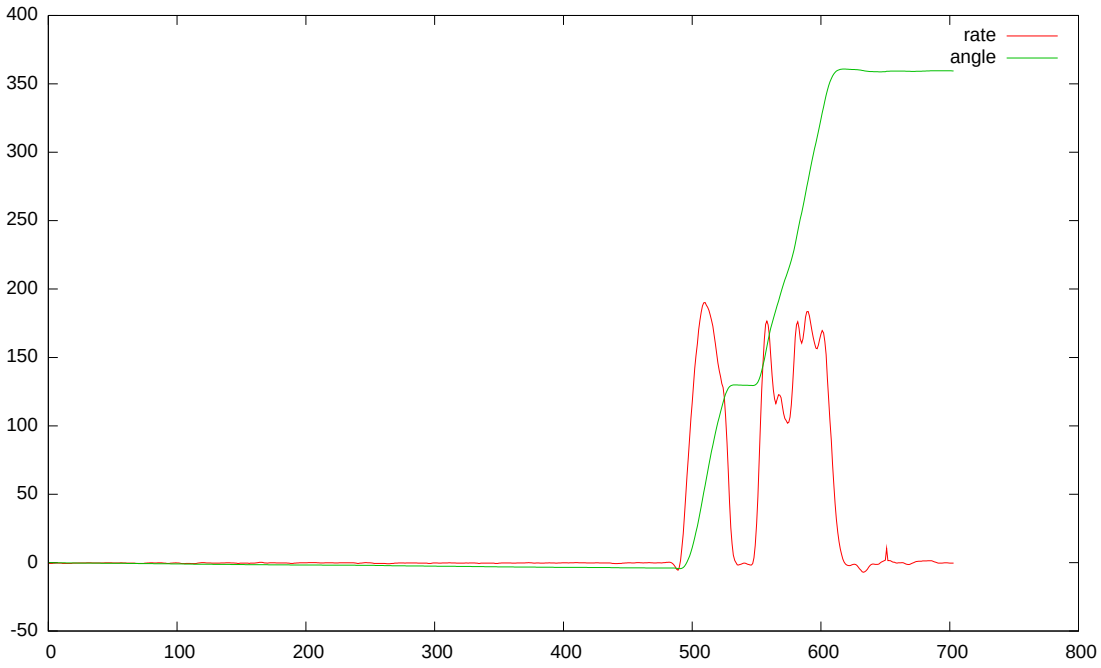


Figure 5.2: Zero offset measurement (raw and filtered)

5.3 Testing in real environment

First we did static testing of the sensors. They were placed on the robot and the drift was measured. We have captured data over 1 minute period. These results have been plotted separately for accelerometer and gyroscope. Accelerometer drift can be seen in figure (5.3). The red and green lines represent the measured values (scaled 100times) and cyan and violet lines represent the integrated speed from the acceleration. It seems that the drift is about $0.002g$ for both X and Y axis, this drift

results in $0.25m/s$ speed difference over one minute of time, which is about $6m$ off from the start position in both X and Y axis.

Gyroscope data have been plotted in figure (5.4). Green line shows the rate measured by the sensor which was scaled 10 times, red line represents integrated angle. Drift of this angle is about 10 degrees per minute.

Both sensors have some drift and cannot by used for long term observations, this has been suspected on the beginning and by doing these tests we were proven right.

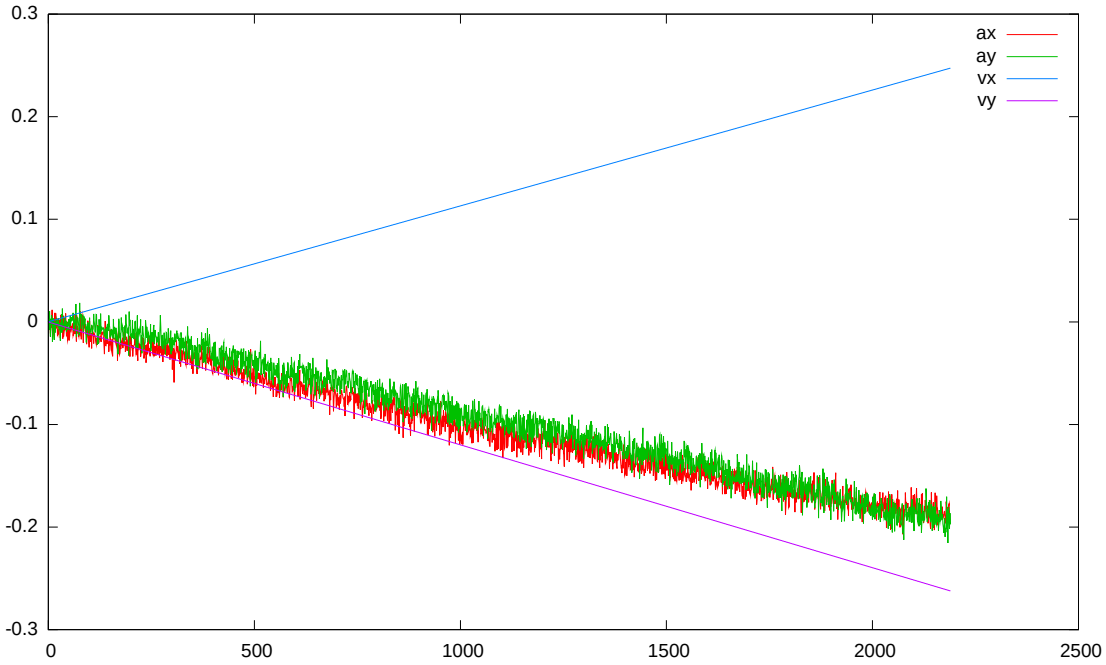


Figure 5.3: Accelerometer drift

Next test we have done is moving the robot half meter forwards. We are trying to test the acceleration sensor, and its response to movement. The accelerometer measurement and the calculated speed can be seen in figure (5.5). We can clearly see the peak at about $x = 5$ on the accelerometer X-axis (green line), this peek shows us that robot has started to move forward. There is a negative peak at about $x = 65$ indicating that robot has came to stop, this causes the integrated speed to decrease. But the measured deceleration it not enough to zero our speed. This might be caused by the sensor not being sensitive enough to small changed, or the small changes do get filtered out by Kalman filter. We can also see that at the end of the movement there is some acceleration present. This is probably caused by the test platform not being leveled perfectly.

Gyroscope was tested while the robot has been doing 90 degree forward turn. The recorded data can be seen in figure (5.6). Green line represents the actual angular speed, and red line represents calculated angle. As we can see the performance of the gyroscope is outstanding in this example.

Another similar example was recorded, while the robot has been doing 180 degree turn (figure 5.7). Again the gyroscope has provided us with precise information about the angle. There is very little to none noise in the gyroscope angular rate reading.

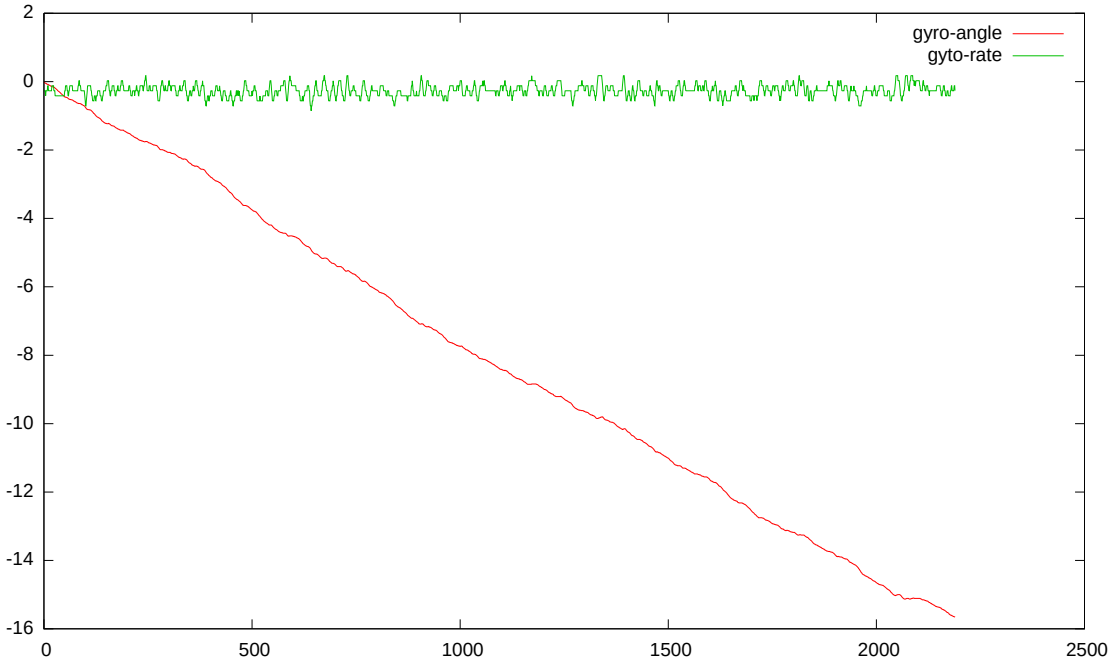


Figure 5.4: Gyroscope drift

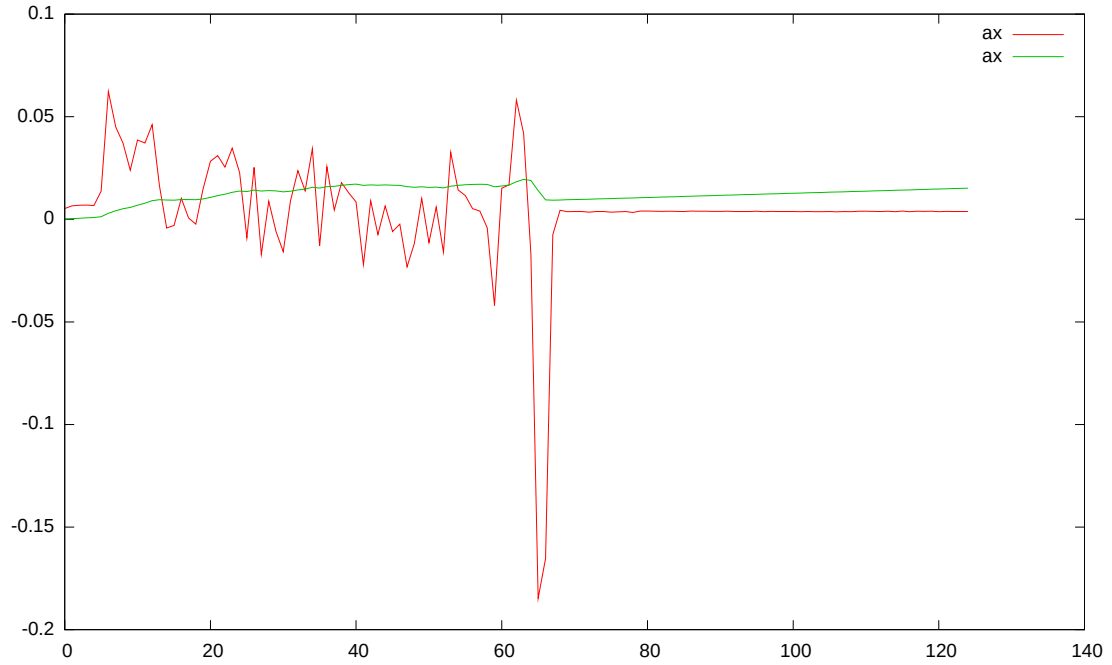


Figure 5.5: Fowrard move with stop

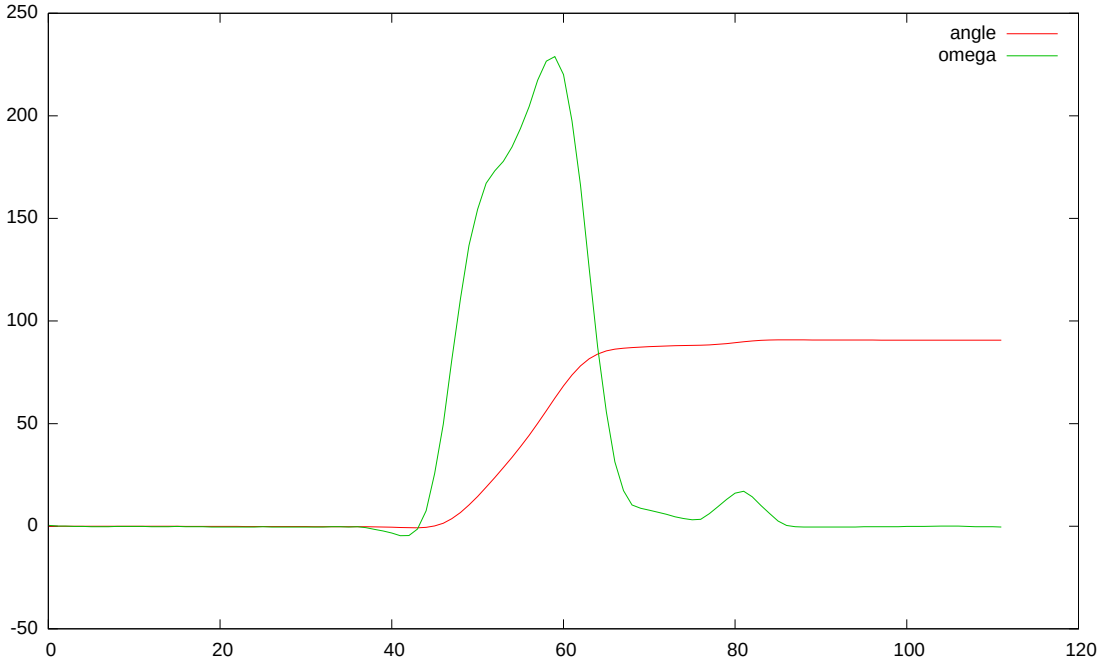


Figure 5.6: 90 degree forward turn

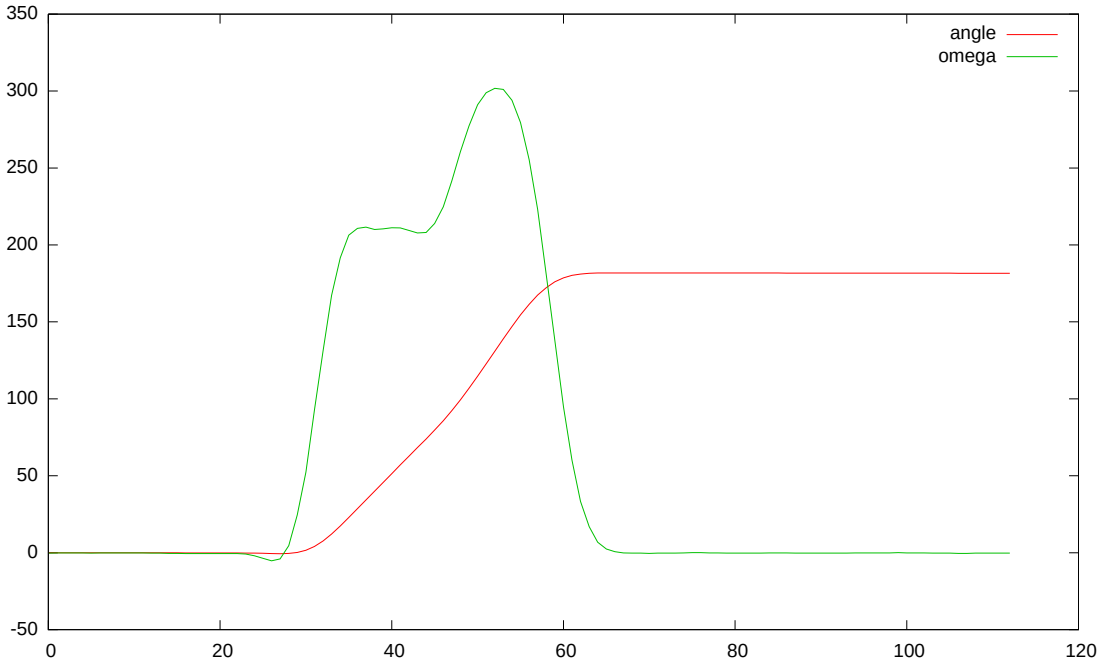


Figure 5.7: 180 degree forward turn

Both sensors, acceleration and gyroscope have been tested. These tests have provided us with valuable information about further selection of sensors for localization of autonomous robot.

Chapter 6

Conclusion

The purpose of this work was to design localization of autonomous robot using MEMS sensors. We have created noise resistant relative robot localization to be used with MEMS sensors. This implementation is platform independent, and configurable for embedded systems. For deeply embedded applications we have implemented Fixed-point arithmetics, which is optimal compromise between precision and speed.

Selected method of filtering data have provided outstanding performance with noise filtering. Tests show that most of the measurement noise has been eliminated.

Alignment method shows that any tilt of the acceleration sensor can be compensated, even when the tilt cannot be seen by bare eye. The method using quaternions is less intuitive for humans, yet more effective for computer calculations as it uses less operations.

Developed calibration methods are precise enough for use in navigation. These calibrations methods are useless for factory calibrated sensors, which are already available on the market.

Despite our efforts we were unable to eliminate all measurement noise and offsets from the acceleration data. These sensors have proved unsuitable for this kind of work and other means of measuring speed should be considered. These sensors are highly sensitive to tilt on non leveled platforms that robot is used on. Using such sensors can lead to major problems with localization with possibly fatal results.

The MEMS sensors drift can cause serious errors in long term position observations. The drift can be "reset" by reseting some of the position parameters and overriding values with known more precise measurements. Other sensors, such as odometers, can be used to provide precise data.

For better performance Extended Kalman Filter should be implemented. This also includes modeling the movement model better. Providing missing degrees of freedom to the IMU - tilt/pan, would help to sense the movement in 3D. This would also help to sense accelerometer tilt/pan provide compensation for it. This would eliminate offsets from the accelerometer sensor, improving the overall accuracy of speed measurement.

Another sensor of velocity should be considered to replace the accelerometer completely. Such sensors are odometers, or range finders. Odometers could provide us with precise velocity information with better noise immunity. Such information

would not suffer from the integration error described before. Resulting in superior performance over acceleration sensor. Range finders could be used on small field to detect edges. If known edge would be sensed, we can use the rotation parameter and some predictions to calculate position on the field.

Appendix A

Library interface

Each sensor of the library has own structure that contains measurement data, calibration data. For each sensor we have also created structure to store raw measurements.

For accelerometer these structures are

```
struct accel_raw_t
struct accel_t
```

For gyroscope

```
struct gyro_raw_t
struct gyro_t
```

Hardware dependent function should be provided by user, that will fill the corresponding *_raw_t structures with raw sensor measurements. Before sensors structures can be used they need to be initialized. For initialization we have provided *_init functions.

```
void accel_init(struct accel_t *);
void gyro_init(struct gyro_t *);
```

After the raw measurements have been acquired we need to process them to create usable data. Each sensor provides *_process function that does normalization, filtration (and alignment for accelerometer).

```
void accel_process(struct accel_t *, struct accel_raw_t *);
void gyro_process(struct gyro_t *, struct gyro_raw_t *);
```

All the parameters (filter, normalization, alignment) can be changed by directly accessing the members of the structure. To initialize Kalman filter with different parameters kalman_init function should be used.

```
void kalman_init(struct kalman_t*, frac Q, frac R);
```

Positioning subsystem is standalone, and needs to be initialized separately. All the position parameters are stored in separate structure.

```
struct position_t
```

As with the other structures, positioning structure should be initialized before using by calling the position_init function. This ensures that all parameters are cleared and that the time difference between measurements is set properly.

```
void position_init(struct position_t *, frac dt);
```

After proper initialization the position subsystem can be fed with sensor data. This data has to be provided in separate structure

```
struct mems_sensors_t
```

This structure has both sensor structures "wrapped" for simpler manipulation.

To feed the position system with sensor data position_process_measurement can be used. We have also provided interface to override various values when more precise measurements are available. After all the correct values are in the library user needs to call position_update function, that updates the position parameters to represent the current location.

```
void position_process_measurement
        (struct position_t *, struct mems_sensors_t *sens);
void position_process_angle
        (struct position_t *, frac omega);
void position_process_accel
        (struct position_t *, frac iax, frac iay);
void position_process_speed
        (struct position_t *, frac ivx, frac ivy);
void position_update(struct position_t *);
```

Appendix B

Usage example

Purpose of this example is to show how to use the library to process actual sensor data. Function data_capture is used to capture data from IMU and put the measurements in corresponding structures - Accelerometer data into accel_raw, Gyroscope data into gyro_raw. This function is not part of the library implementation and should by provided by user. We have created reference implementation that has also been used to test the functionality.

After the sensor data was put into correct place, we can call *_process functions. These functions take care of filtering and applying normalization/calibration. It is assumed that the sensor structures have been filed with correct values before these functions are executed. Failure to do so may result in random position to be calculated.

Next step is to calculate position from the provided data. We are updating all the sensor parameters at once using position_process_measurements. This function inserts fresh measurements into the position structure. To calculate the actual position in this time-step position_update function needs to be called.

At last we print the calculated position to standard output.

```
while (1)       {
  struct accel_raw_t accel_raw;
  struct gyro_raw_t gyro_raw;

  data_capture(dev_fd, &accel_raw, &gyro_raw);

  // apply normalization data
  accel_process(&mems_sensors.accel, &accel_raw);

  // apply normalization on gyro
  gyro_process(&mems_sensors.gyro, &gyro_raw);

  // calculate position
  position_process_measurement(&position, &mems_sensors);
  position_update(&position);
  position_print(&position);
}
```

Appendix C

CD content

- HW
  - dsPIC30F6011A - HW firmware used to capture data samples
- SW
  - C - example used to test library implementation and calibration routines
    - MEMS - Localization library implementation
    - data - various movement logs and gnuplot scripts to generate graphs
    - logs - raw sensor measurements
  - Python - test implementation in python
    - Python/cal_data - calibration data used to create graphs
    - Python/gyro_data - gyroscope data and gnuplot scripts, that were used to create graphs
    - Python/accel_data - accelerometer data and gnuplot scripts
- thesis - Latex source of the thesis
  - mems - various pictures used in the thesis
  - 15g - various graphs from accelerometer
  - gyro - various graphs from gyroscope

Bibliography

[1] MEMS and Nanotechnology Exchange, What is MEMS Technology?, http://www.mems-exchange.org/MEMS/what-is.html

[2] STMicroelectronics NV, *LIS3LV02DL*, 2008, www.st.com/stonline/books/pdf/docs/12094.pdf

[3] Analog Devices, Inc., *ADIS16250*, 2009 http://www.analog.com/static/imported-files/Data_Sheets/ADIS16250_16255.pdf

[4] Greg Welch and Gary Bishop, *An Introduction to the Kalman filter* , 2006, http://www.cs.unc.edu/ welch/kalman/

[5] Ken Shoemake, *Quaternions, Department of Computer and Information Science*, University of Pennsylvania

[6] David Eberly, *Quaternion Algebra and Calculus*, Geometric Tools LCC, 1999

[7] Erick Oberstar L., *Fixed Point Representation And Fractional Math*, 2007, http://www.superkits.net/whitepapers.htm

[8] Randy Yates, *Fixed-Point Arithmetic: An Introduction* , Digital Signal Labs, 2009, www.digitalsignallabs.com/fp.pdf

[9] John Halleck, *Factoring of Integers Rarely Asked Questions*, 2009, http://home.utah.edu/ nahaj/